```
/*
 * File:    IRTemp.cpp
 * Version: 1.0
 * Author:  Andy Gelme (@geekscape) & Angus Gratton <angus at freetronics. com>
 * License: GPLv3
 *
 * See www.freetronics.com/irtemp for more information.
 *
 * ToDo
 * ~~~~
 * - Cache previously read Ambient or IR temperature in case get*Temperature()
 *   is called more often than 0.1 seconds (minimum time between updates).
 */

#ifndef IRTEMP_cpp
#define IRTEMP_cpp

#include "IRTemp.h"

static const byte IRTEMP_DATA_SIZE = 5;

static const long IRTEMP_TIMEOUT = 1000;  // milliseconds

// Each 5-byte data packet from the IRTemp is tagged with one of these
static const long IRTEMP_DATA_AMBIENT = 0x66;
static const long IRTEMP_DATA_IR =      0x4C;
//static const long IRTEMP_DATA_JUNK =    0x53; // ignored, contains version info perhaps?

IRTemp::IRTemp(
```

```
  byte pinAcquire,
  byte pinClock,
  byte pinData) {

  _pinAcquire = pinAcquire;
  _pinClock =   pinClock;
  _pinData =    pinData;

  if(_pinAcquire != -1) {
    pinMode(_pinAcquire, OUTPUT);
    digitalWrite(_pinAcquire, HIGH);
  }
  pinMode(_pinClock,   INPUT);
  pinMode(_pinData,    INPUT);

  digitalWrite(_pinClock,   HIGH);
  digitalWrite(_pinData,    HIGH);

  sensorEnable(false);
}

float IRTemp::getAmbientTemperature(
  TempUnit scale) {

  return(getTemperature(scale, IRTEMP_DATA_AMBIENT));
}

float IRTemp::getIRTemperature(
  TempUnit scale) {
```

```
  return(getTemperature(scale, IRTEMP_DATA_IR));
}

float IRTemp::getTemperature(
  TempUnit scale,
  byte dataType) {

  long timeout = millis() + IRTEMP_TIMEOUT;

  sensorEnable(true);

  while(1) {
    uint8_t data[IRTEMP_DATA_SIZE] = { 0 };
    for(uint8_t data_byte = 0; data_byte < IRTEMP_DATA_SIZE; data_byte++) {
      for(int8_t data_bit = 7; data_bit >= 0; data_bit--) {
        // Clock idles high, data changes on falling edge, sample on rising edge
        while(digitalRead(_pinClock) == HIGH && millis() < timeout) { } // Wait for falling edge
        while(digitalRead(_pinClock) == LOW && millis() < timeout) { } // Wait for rising edge to sample
        if(digitalRead(_pinData))
          data[data_byte] |= 1<<data_bit;
      }
    }
    if(millis() >= timeout) {
      sensorEnable(false);
      return NAN;
    }

    if (data[0] == dataType && validData(data)) {
```

```
    sensorEnable(false);
    float temperature = decodeTemperature(data);
    if (scale == FAHRENHEIT)
      temperature = convertFahrenheit(temperature);
    return temperature;
   }
  }
}

float IRTemp::convertFahrenheit(
  float celcius) {

  return(celcius * 9 / 5 + 32);
}

float IRTemp::decodeTemperature(
  volatile byte data[]) {

  int msb = data[1] << 8;
  int lsb = data[2];

  return((msb + lsb) / 16.0 - 273.15);
}

void IRTemp::sensorEnable(
  bool state) {
  if(_pinAcquire != -1)
    digitalWrite(_pinAcquire, ! state);
}
```

```
bool IRTemp::validData(
  byte data[]) {

  byte checksum = (data[0] + data[1] + data[2]) & 0xff;

  return(data[3] == checksum  &&  data[4] == '\r');
}

#endif
```